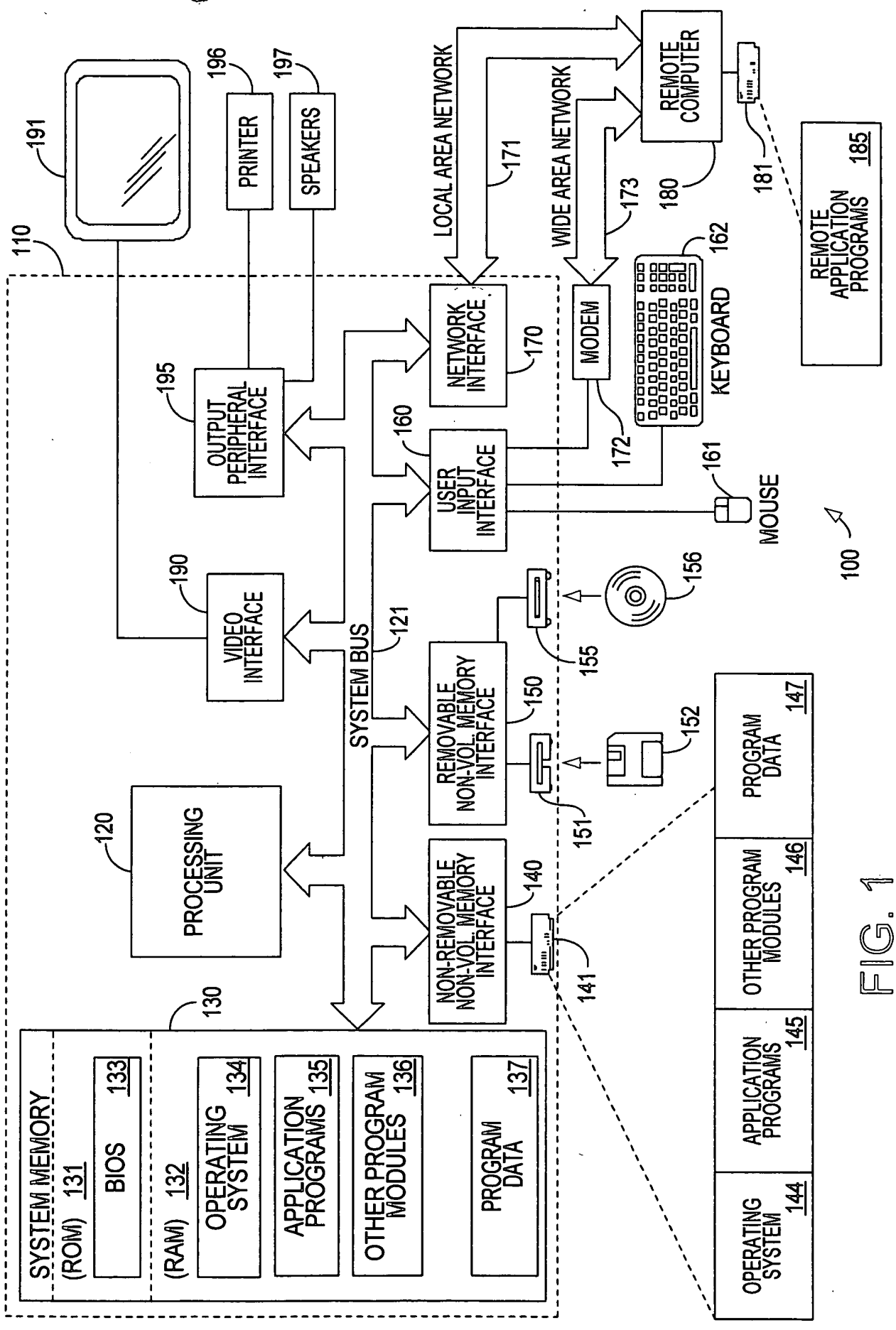

SYSTEM MEMORY
(ROM) 131
BIOS 133
(RAM) 132
OPERATING SYSTEM 134
APPLICATION PROGRAMS 135
OTHER PROGRAM MODULES 136
PROGRAM DATA 137
130
PROCESSING UNIT
120
SYSTEM BUS
121
VIDEO INTERFACE
190
OUTPUT PERIPHERAL INTERFACE
195
NETWORK INTERFACE
170
USER INPUT INTERFACE
160
REMOVABLE NON-VOL. MEMORY INTERFACE
150
NON-REMOVABLE NON-VOL. MEMORY INTERFACE
140
MODEM
172
KEYBOARD
162
MOUSE
161
PRINTER
196
SPEAKERS
197
191
LOCAL AREA NETWORK
171
WIDE AREA NETWORK
173
REMOTE COMPUTER
180
181
REMOTE APPLICATION PROGRAMS
185
155
156
151
152
141
100
OPERATING SYSTEM 144
APPLICATION PROGRAMS 145
OTHER PROGRAM MODULES 146
PROGRAM DATA 147

FIG. 1

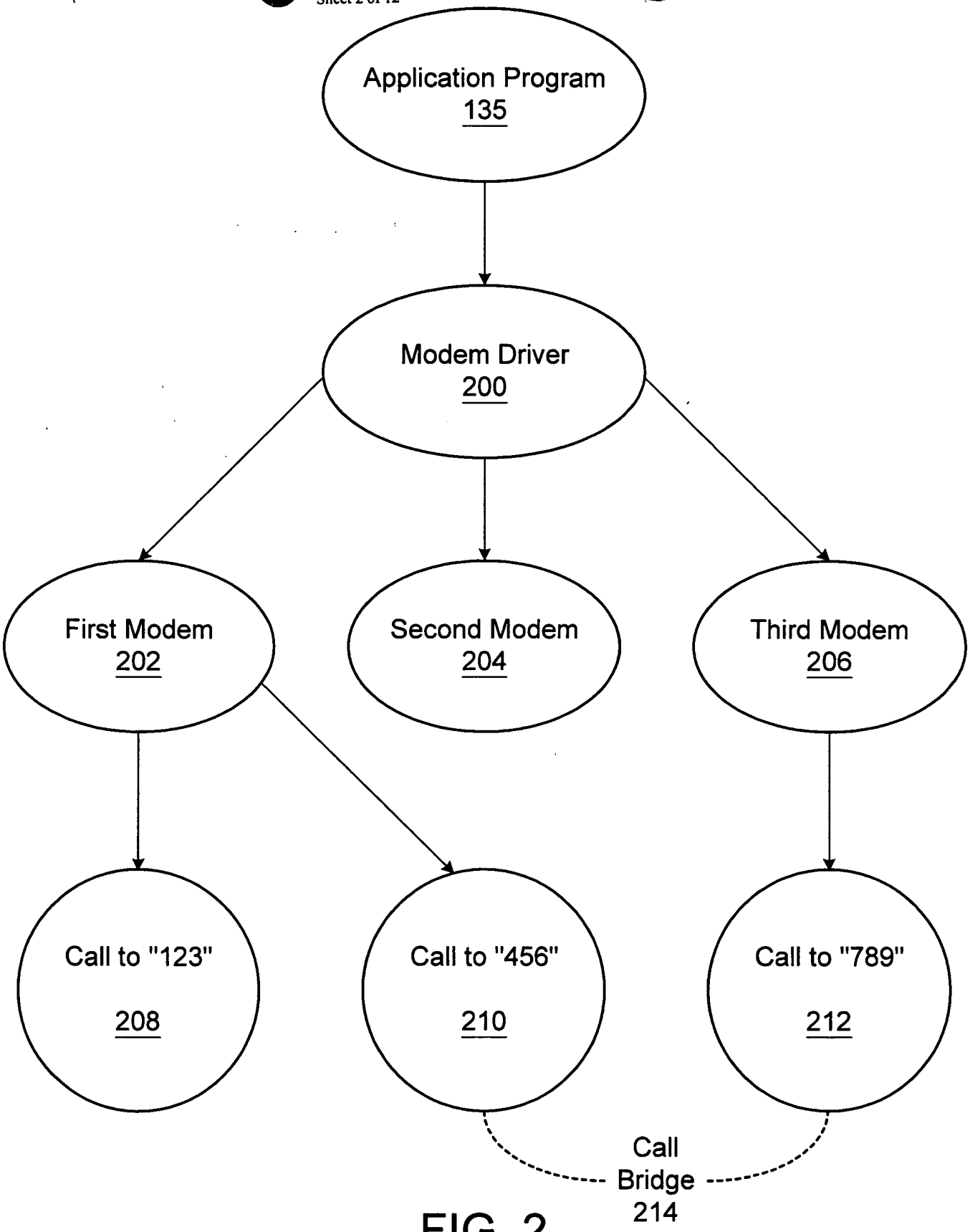

Application Program
135
Modem Driver
200
First Modem
202
Second Modem
204
Third Modem
206
Call to "123"
208
Call to "456"
210
Call to "789"
212
Call
Bridge
214

FIG. 2

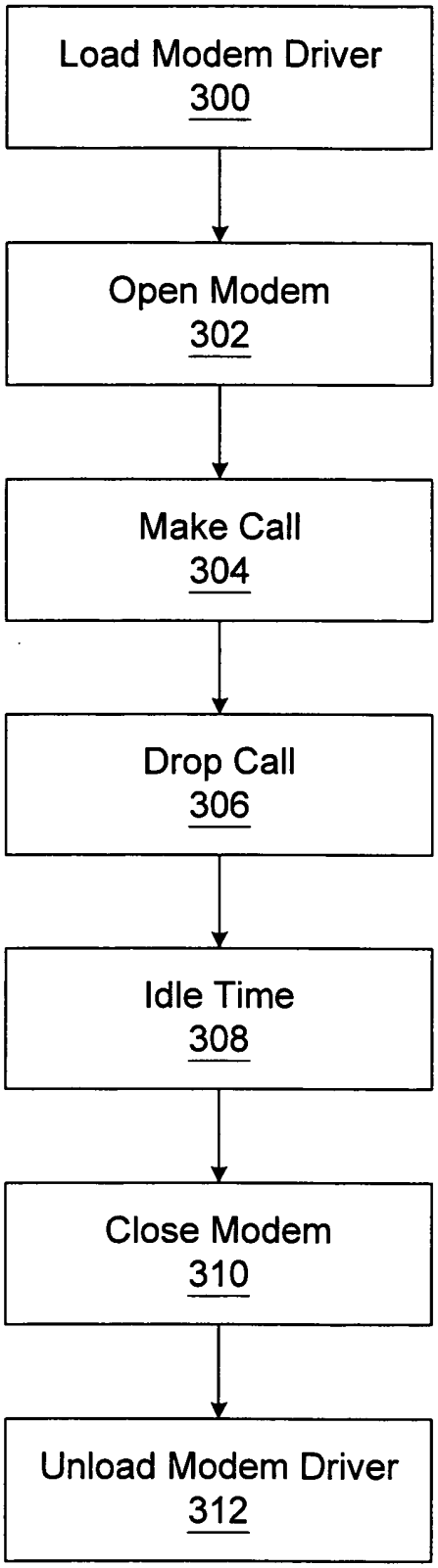

Load Modem Driver
300
Open Modem
302
Make Call
304
Drop Call
306
Idle Time
308
Close Modem
310
Unload Modem Driver
312

FIG. 3A

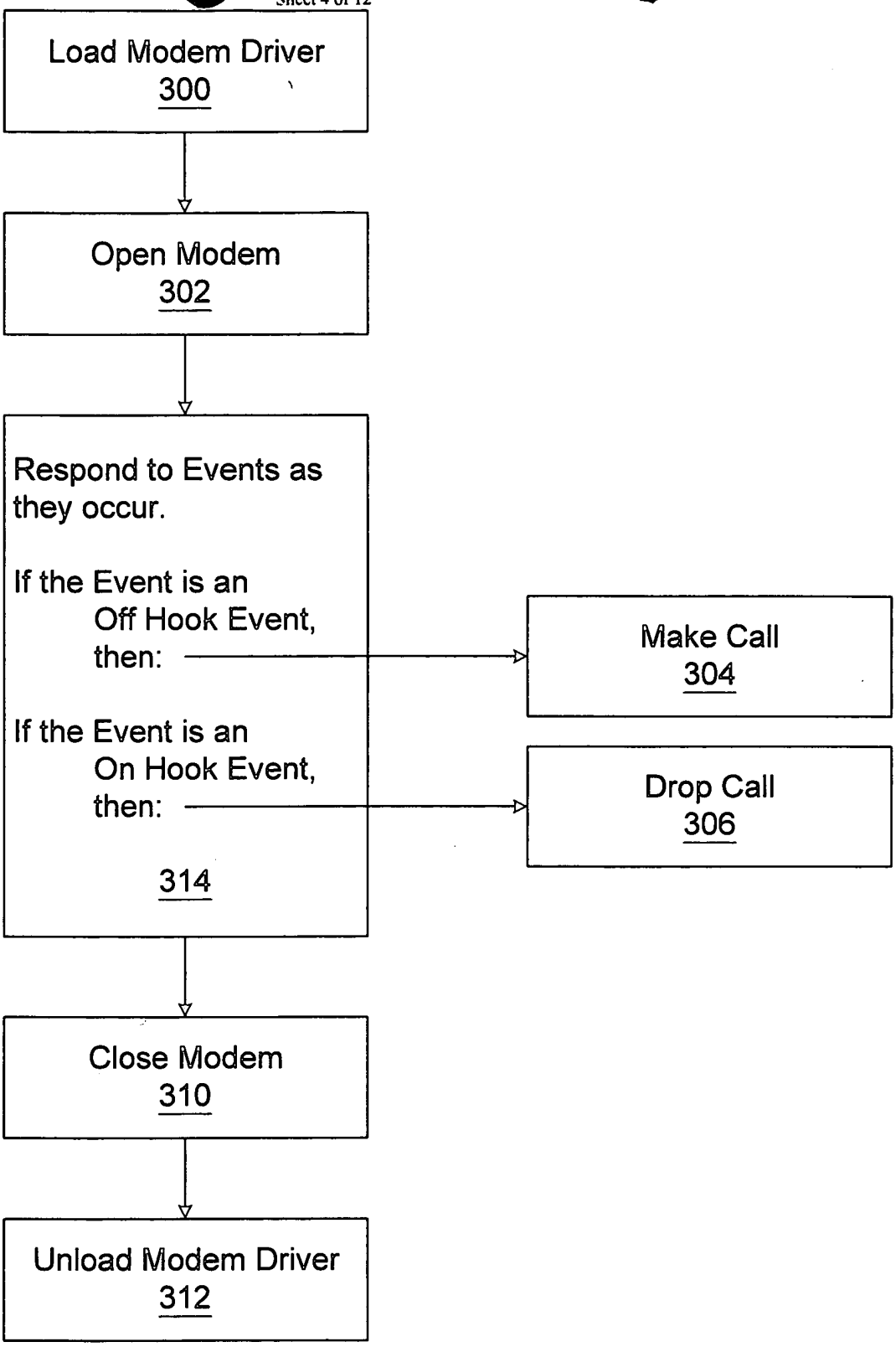

Load Modem Driver
300
Open Modem
302
Respond to Events as they occur.
If the Event is an Off Hook Event, then:
If the Event is an On Hook Event, then:
314
Make Call
304
Drop Call
306
Close Modem
310
Unload Modem Driver
312

FIG. 3B

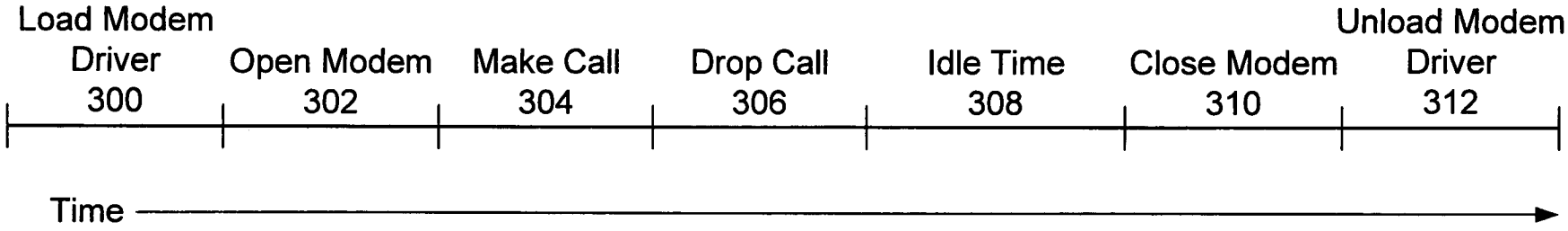


FIG. 4A: Synchronous Process Model

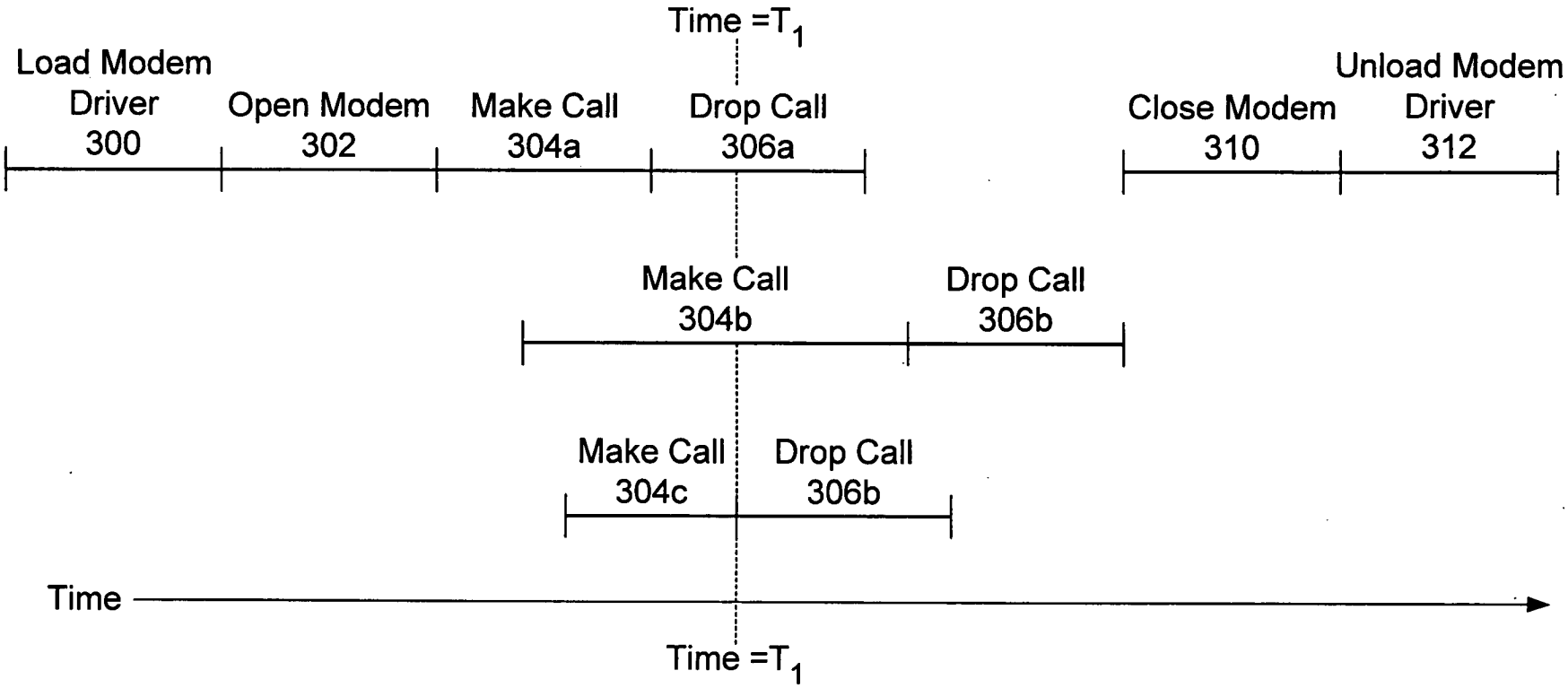


FIG. 4B: Asynchronous Process Model

FIG. 5

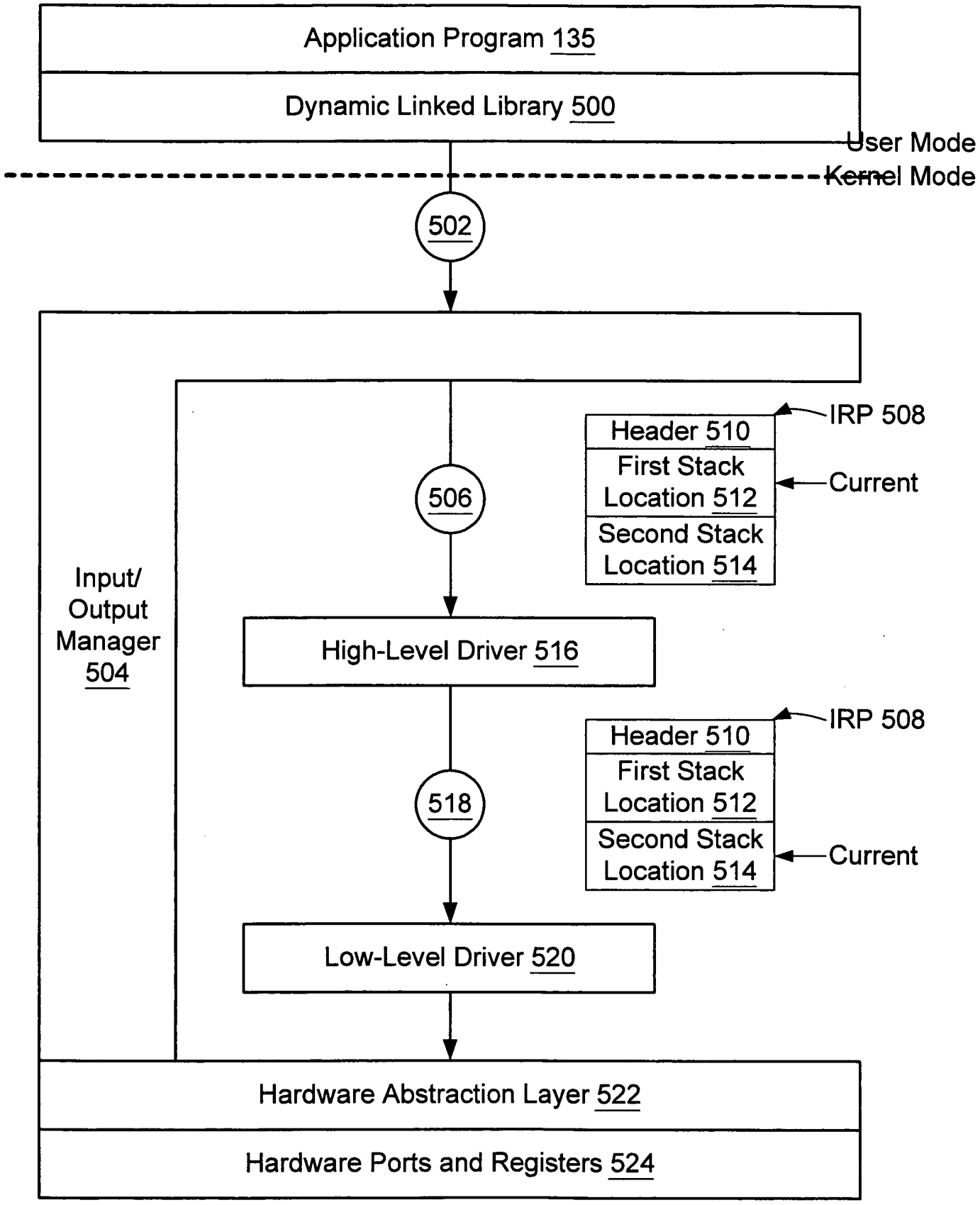

Application Program 135
Dynamic Linked Library 500
User Mode
Kernel Mode
502
Input/ Output Manager 504
506
IRP 508
Header 510
First Stack Location 512
Current
Second Stack Location 514
High-Level Driver 516
518
IRP 508
Header 510
First Stack Location 512
Second Stack Location 514
Current
Low-Level Driver 520
Hardware Abstraction Layer 522
Hardware Ports and Registers 524

FIG. 6

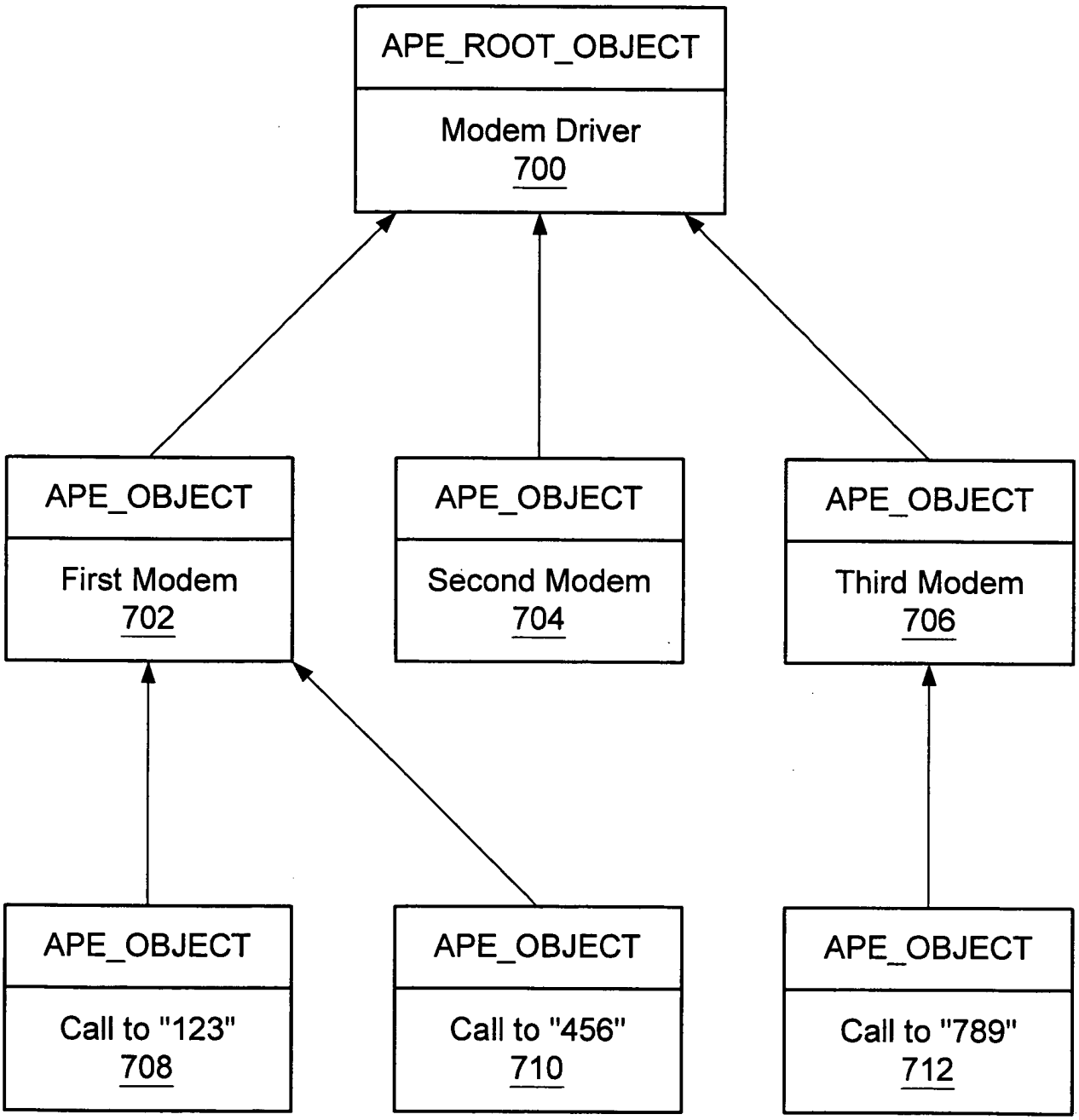
```
BOOL DoCallSync(UINT ModemNo, TCHAR *tszDialString)
{
        HMODEM          hModem;
        HCALL           hCall;
        STATUS          Status;

        LoadDriver();
        Status = OpenModem(ModemNo, NULL, NULL, &hModem);
        if(Status == SUCCESS)
        {
                Status = MakeCall(hModem, tszDialString, NULL, &hCall);
                if(Status == SUCCESS)
                {
                        DropCall(hCall);
                }

                CloseModem(hModem);
        }

        UnloadDriver();
        return Status == SUCCESS;
}
```

FIG. 7


APE_ROOT_OBJECT
Modem Driver
700
APE_OBJECT
First Modem
702
APE_OBJECT
Second Modem
704
APE_OBJECT
Third Modem
706
APE_OBJECT
Call to "123"
708
APE_OBJECT
Call to "456"
710
APE_OBJECT
Call to "789"
712

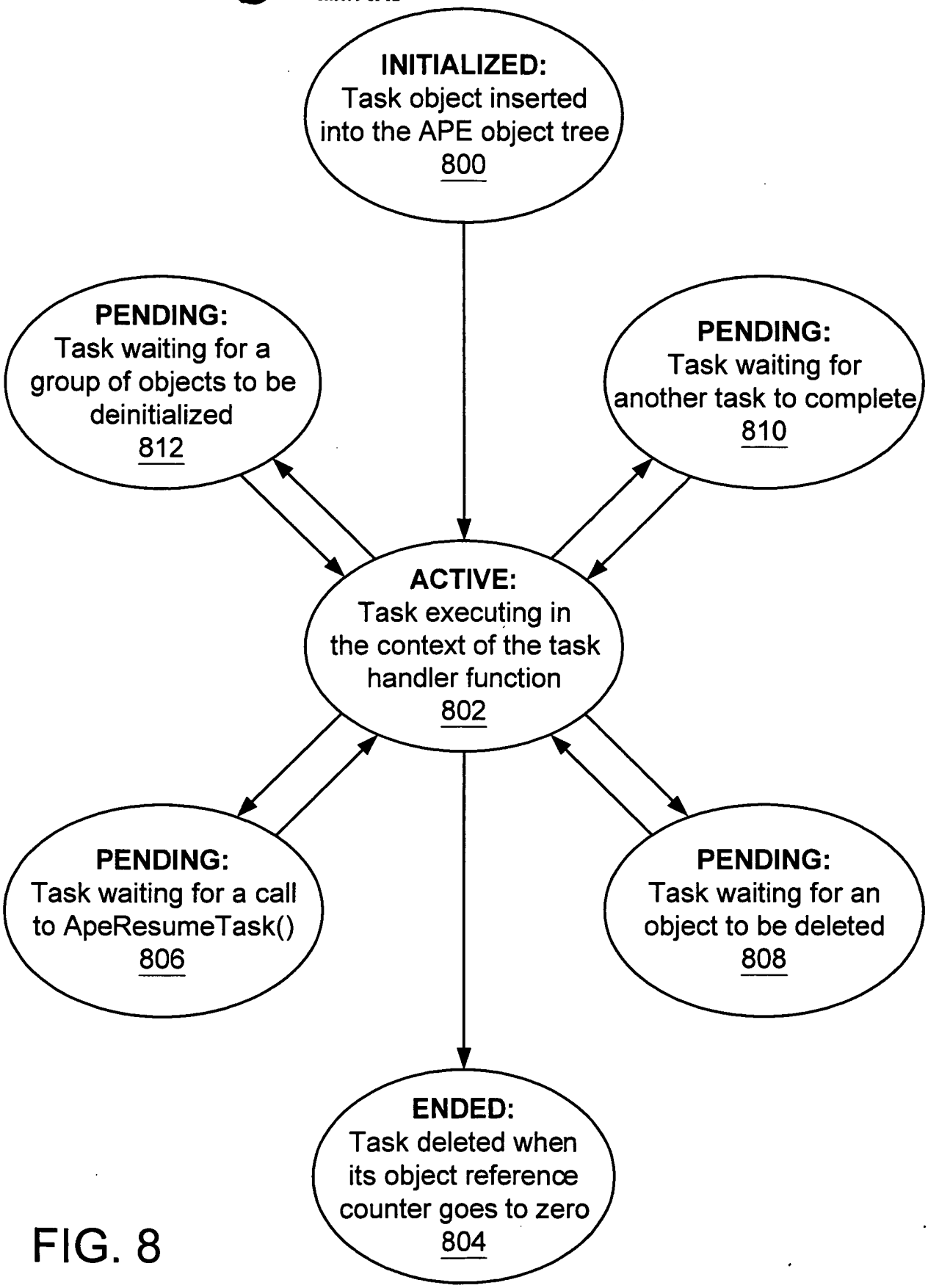

INITIALIZED:
Task object inserted into the APE object tree
800
PENDING:
Task waiting for a group of objects to be deinitialized
812
PENDING:
Task waiting for another task to complete
810
ACTIVE:
Task executing in the context of the task handler function
802
PENDING:
Task waiting for a call to ApeResumeTask()
806
PENDING:
Task waiting for an object to be deleted
808
ENDED:
Task deleted when its object reference counter goes to zero
804

FIG. 8

FIG. 9

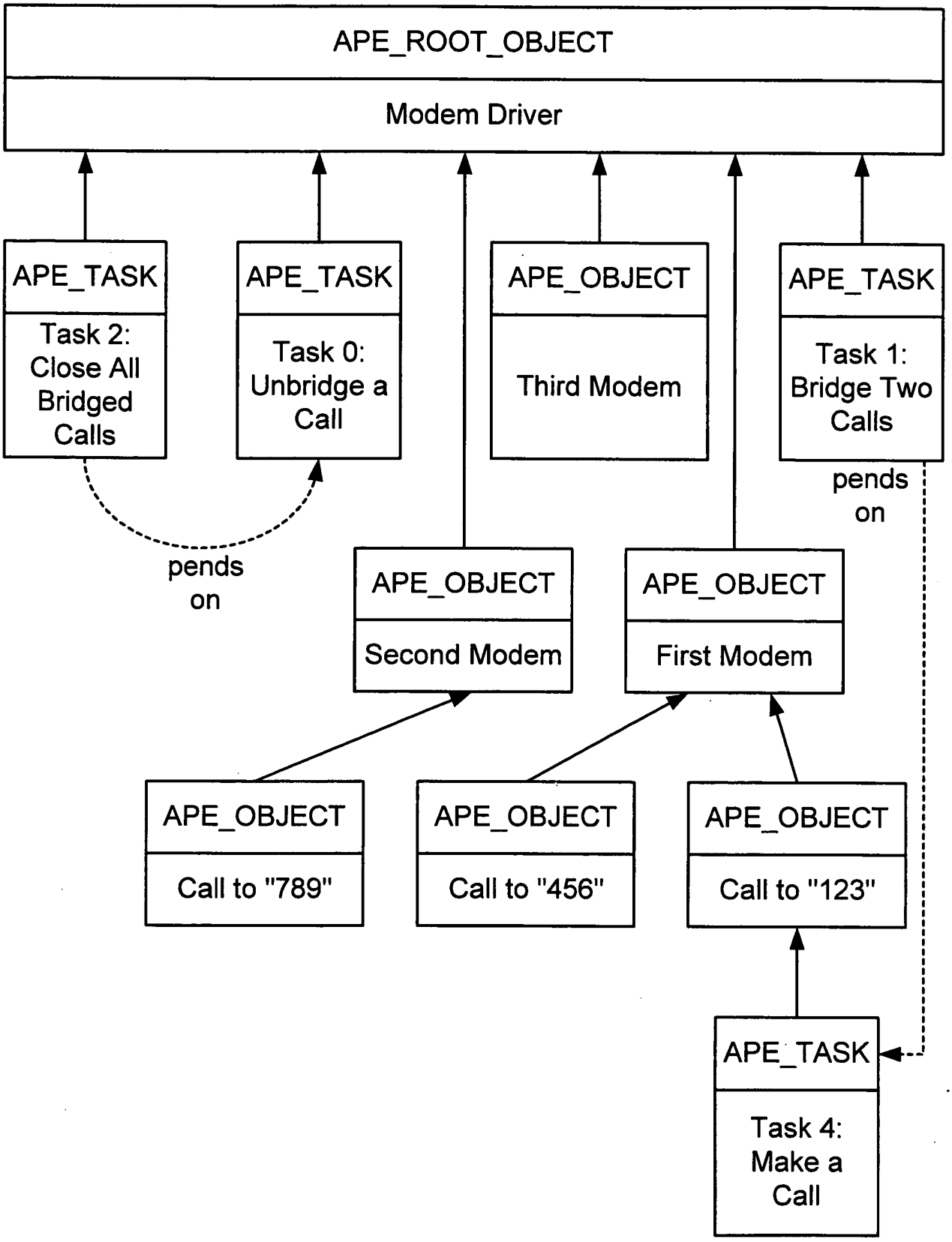

APE_ROOT_OBJECT
Modem Driver
APE_TASK
Task 2: Close All Bridged Calls
APE_TASK
Task 0: Unbridge a Call
APE_OBJECT
Third Modem
APE_TASK
Task 1: Bridge Two Calls
pends on
pends on
APE_OBJECT
Second Modem
APE_OBJECT
First Modem
APE_OBJECT
Call to "789"
APE_OBJECT
Call to "456"
APE_OBJECT
Call to "123"
APE_TASK
Task 4: Make a Call

FIG. 10

```
VOID DoCallAsync
(
    UINT                        ModemNo,
    TCHAR                       *tszDialString,
    PFN_COMPLETION_HANDLER      pfnCompletionHandler,
    PVOID                       pvCompletionContext
)
{
    APE_DECLARE_STACK_RECORD(SR, pSR, 0xf7fe39d2)
    PDOCALL_TASK      pTask;

    pTask =
        AllocateDoCallTask
        (
            ModemNo, tszDialString, pfnCompletionHandler,
            pvCompletionContext, pSR
        );
    if(pTask == NULL)
    {
        // Failed to allocate task.  Call completion handler.
        pfnCompletionHandler(pvCompletionContext, FALSE);
    }
    else  // Start the task just allocated.  The task does the actual work.
        ApeStartTask(&pTask->TskHdr, 0x5c5d5ba9, pSR);
}
```

FIG. 11

```
VOID DoCallTask
(
    IN   PAPE_TASK          pTask,
    IN   PAPE_STACK_RECORD  pSR
)
{
    ...
    pState = &pDoCallTask->TskHdr.Hdr.UserState;
    switch(*pState)
    {
        case START:
            //Load the modem driver. This is a synchronous call.
            MdmLoadDriver();

            // Open the modem. This is an asynchronous call.
            MdmStatus =
                MdmOpenModem
                (
                    pDoCallTask->ModemNo,
                    ModemCompletionHandler, pDoCallTask,
                    &pDoCallTask->hModem
                );
            if(MdmStatus == MDM_STATUS_PENDING)
            {
                // Suspend this task and resume when
                // MdmOpenModem() completes asynchronously.
                *pState = OPENMODEM;
                ApeSuspendTask(pTask, pSR);
                break;
            }
            else  // Save the return status here.
                pDoCallTask->AsyncMdmStatus = MdmStatus;

            // Fall through on synchronous completion.
        case OPENMODEM:
            ...
    }
}
```